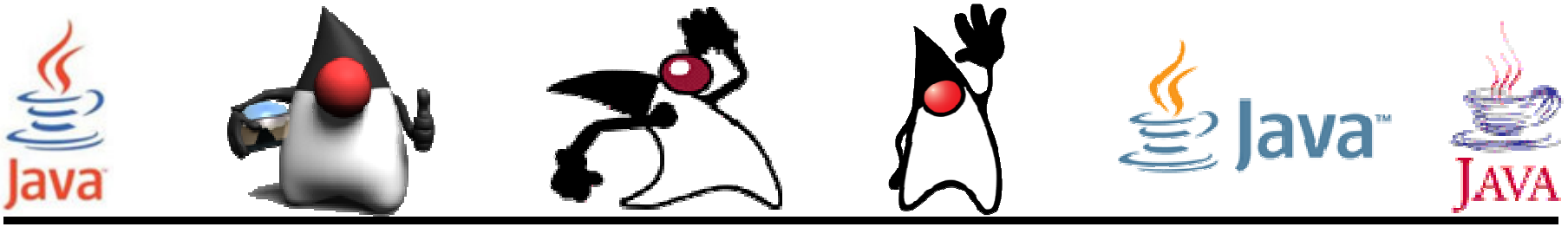

Serialization

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Comprender la serialización de objetos para obtener la persistencia de los mismos
- Estudiar los mecanismos de serialización y control de versión

Índice

- Concepto de Serialización
- Campos preservados en un objeto serializado
- Proceso de serialización
- Proceso de deserialización
- Control de version

Concepto de Serialization

- Habilidad de leer o escribir un objeto en un stream
  - Proceso de “empaquetado” de un objeto
- Se usa para almacenar objetos en memoria permanente
  - el estado se escribirá de forma serializada a un fichero tal que un objeto puede ser reconstruído posteriormente desde ese fichero
- Usado para pasar a otro objeto mediante la clase OutputStream
  - se puede enviar a través de la red

Streams usados para Serialization

- ObjectOutputStream
  - Para la serialización (aplanado de un objeto)
- ObjectInputStream
  - Para la deserialización (reconstrucción de un objeto)

Requerimientos para Serialization

- Para que un objeto sea serializable:
  - Su clase debe implementar la interface *Serializable*
  - Su clase debe proporcionar un constructor por defecto (un constructor sin argumentos)
- La característica de serializabilidad se hereda
  - No tiene que implementar *Serializable* en cada clase
  - Sólo se puede implementar Serializable una vez a la jerarquía de clase

Objetos No Serializables

- La mayoría de clases Java son serializables
- Los objetos de algunas clases a nivel de sistema no son serializables (thread por ejemplo)
- Se lanza un *NotSerializableException* si se intenta serializar un objeto no serializable

Elementos preservados cuando se serializa un objeto

- Suficiente información necesaria para reconstruir la instancia de un objeto en un momento posterior
  - Sólo se preserva los datos del objeto
  - Los métodos y constructores no forman parte del stream serializado
  - Se incluye la información de la clase
- Se puede serializar un objeto que contiene como campos clases no serializables con la palabra reservada *transient* en la declaración del objeto

Proceso de serialización: escribiendo un object stream

- Se usa el método *writeObject* de la clase *ObjectOutputStream*

```
public final void writeObject(Object obj)
throws IOException
```

donde:

obj es el objeto a escribirse en el stream

Ejemplo de serialización: escribiendo un object stream

```
import java.io.*;
public class SerializeBoolean {
  SerializeBoolean() {
    Boolean booleanData = new Boolean("true");
      try {
        FileOutputStream fos = new
                               FileOutputStream("boolean.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(booleanData);
        oos.close();
      } catch (IOException ie) {
        ie.printStackTrace();
      }
  }

  public static void main(String args[]) {
     SerializeBoolean sb = new SerializeBoolean();
  }
}
```

Proceso de deserialización: leyendo un object stream

- Se usa el método *readObject* de la clase *ObjectInputStream*

```
public final Object readObject() throws
  IOException, ClassNotFoundException
```

  donde:

  obj es el objeto a ser leído del stream

- El tipo *Object* retornado debe aplicarse un cast con el nombre de la clase apropiada antes que los métodos de la clase se ejecuten

Ejemplo de deserialización: lectura de un object stream

```
import java.io.*;
public class UnserializeBoolean {
  UnserializeBoolean() {
    Boolean booleanData = null;
      try {
        FileInputStream fis = new FileOInputStream("boolean.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        booleanData = (Boolean)ois.readObject(); ois.close();
      } catch (Exception e) { e.printStackTrace(); }
      System.out.println(“Boolean deserializado de "
                         + "boolean.ser");
      System.out.println(“Datos Boolean: " + booleanData);
      System.out.println("Compara dato con true: " +
                          booleanData.equals(new Boolean("true")));
  }
  public static void main(String args[]) {
     UnserializeBoolean usb = new UnserializeBoolean();
  }
}
```

Control de versión

- Considerando que se crea una clase, se instancia y serializa (mediante un object stream) en un fichero que no se toca por un tiempo.
- Si se actualiza el fichero de clase, por ejemplo añadiendo un nuevo campo, y si se intenta leer el objeto serializado se obtendrá una excepción java.io.InvalidClassException
- Todas las clases con capacidad de persistencia se les asigna automáticamente un identificador único
- La causa de la excepción es que el id no coincide

Control de versión

- Para solventar ese problema hay que proporcionar el campo manualmente y asegurarse que siempre sea el mismo, sin importar los cambios que se realicen en el classfile
- Para generar un único ID, el entorno Java dispone de la utilidad *serialver*

  Ejemplo:
  > serialver MyClass
  MyClass static final long serialVersionUID = 1027553947283749 5L;

Personalización del protocolo de serialización

- Se usa cuando el comportamiento por defecto de *readObject()* y *writeObject()* no son suficientes
- Se puede proporcionar un comportamiento personalizado de *readObject()* y *writeObject()*

Ejemplo:

```
// Metodo readObject propio
private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
   //  "pseudo-constructor“ propio
   in.defaultReadObject();
   // ahora es un objeto “vivo", por lo que se hace un rebuild y start
   startAnimation();
}
```